



# FUNDAMENTAL OF PROGRAMMING IN C#

USING VISUAL STUDIO 2017

# Learning Objectives

- Understand the purpose of IDE and the role of Visual Studio in C# application development
- Write and execute a C# console application using Visual Studio
- Perform step-by-step debugging in Visual Studio

# Agenda

- Integrated Development Environment
- Elements of Visual Studio
- Creating a Solution and Project
- Running your application
- Debugging your application

# Integrated Development Environment (IDE)

- Visual Studio is an IDE
  - Editor
  - Visual Development Tools
  - Compiler/ Builder
  - Debugger
- Purpose
  - Improve developer's productivity by providing various development related features in a single tools
  - Scaffold creation of different types of applications by choosing an appropriate project type
  - Help in packaging the application beyond simple compilation

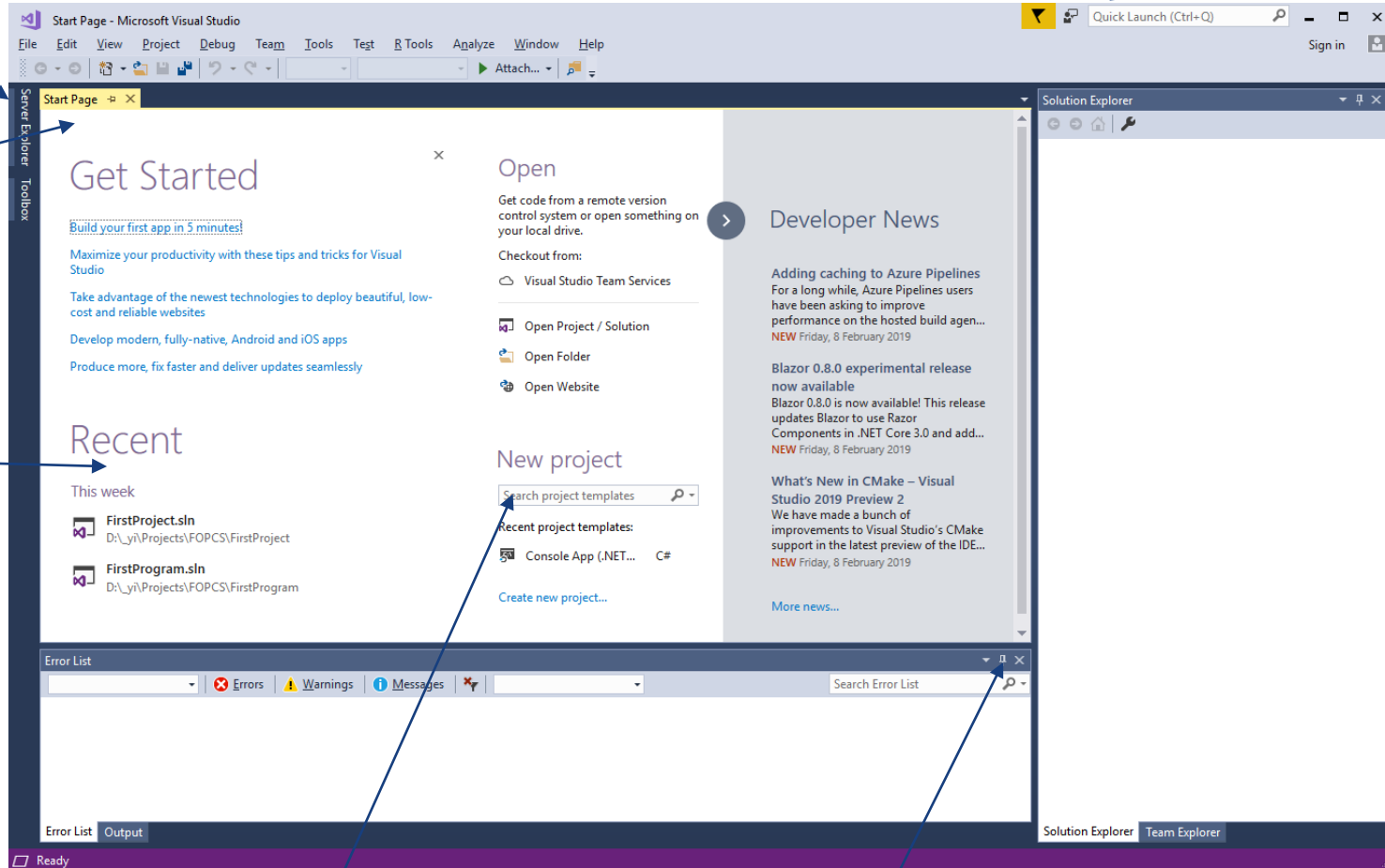
# Visual Studio Screen

Hidden tabs – click to show

Shortcut to launching menu items

Main Window displaying Start Page

Shortcut to your recent projects (very useful)

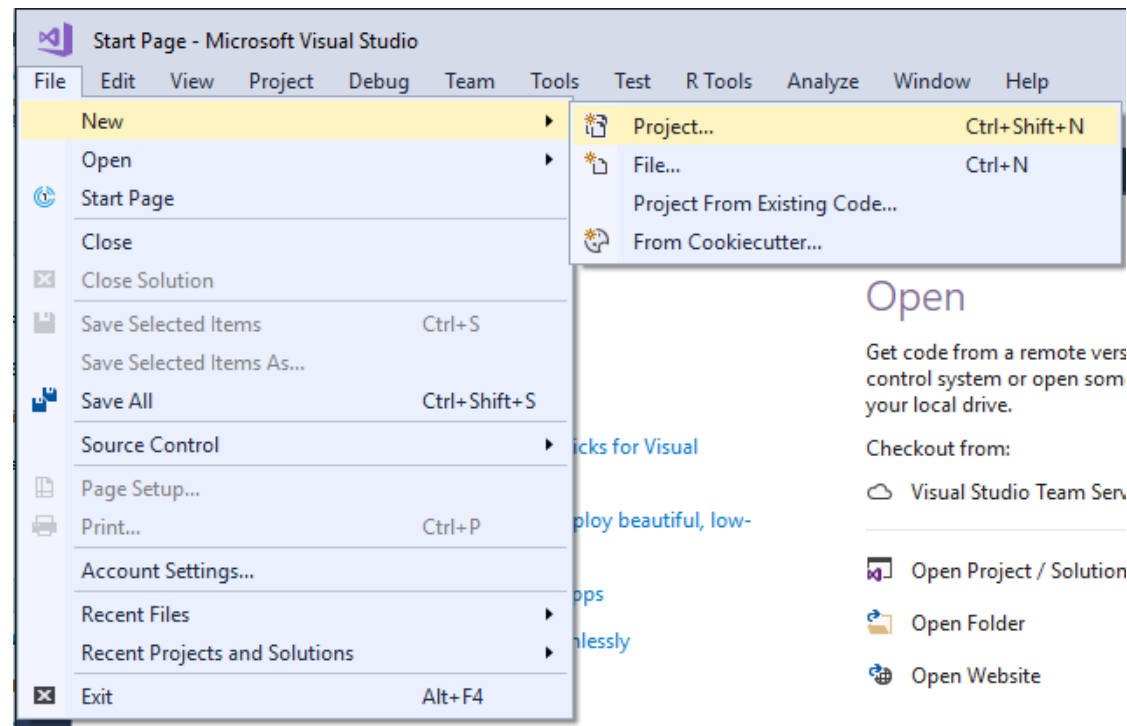


Shortcut for creating a new project

Pin to change the auto hide status – make windows hidden to allow for more space

# Creating your first VS solution

- Multiple ways to create a new solution:
  - use the shortcut on the Start Page,
  - click on File > New > Project
  - Press Ctrl+Shift+N



# Creating your first VS solution

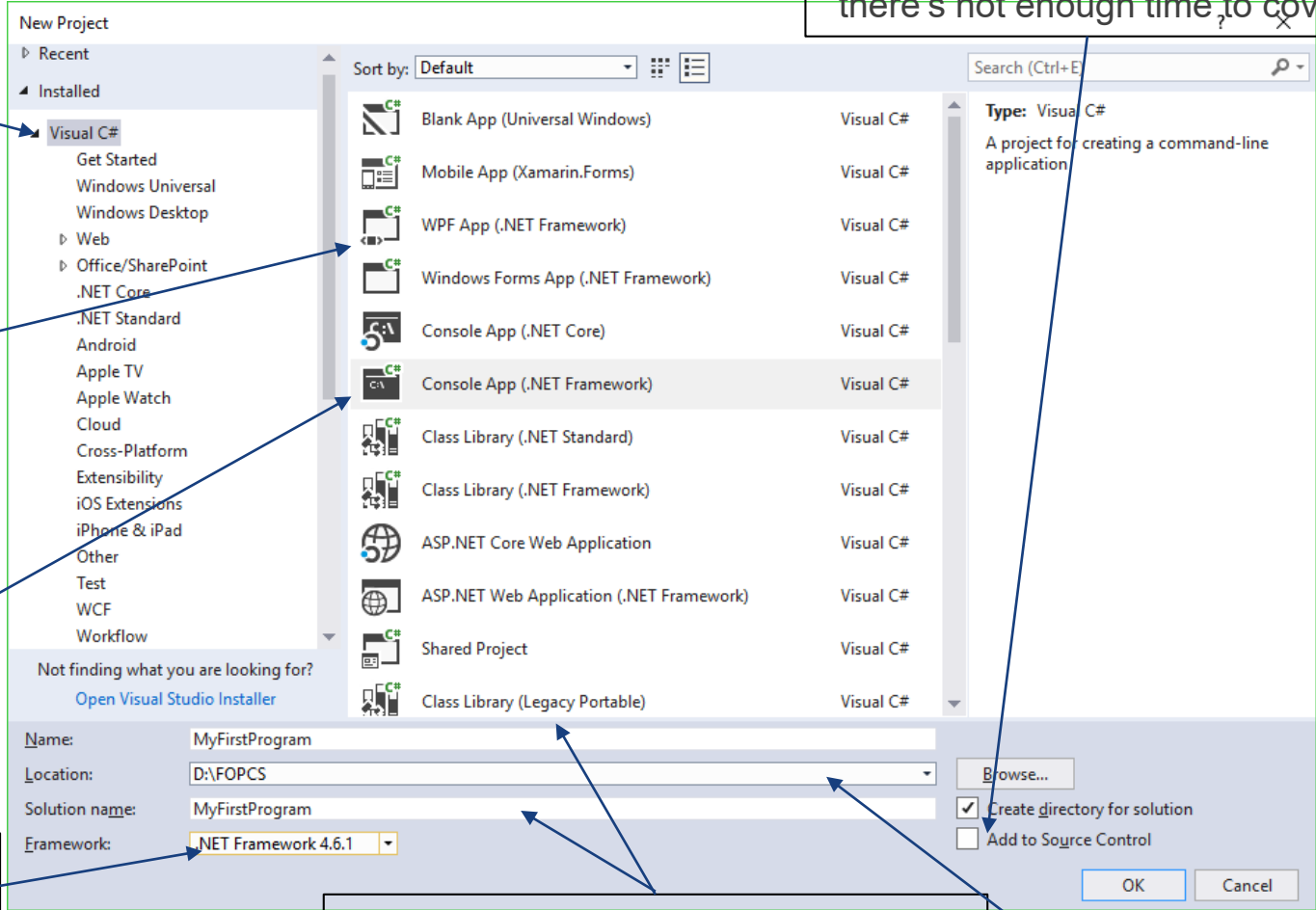
Project Categories grouped by the language – make sure you choose C#

Different type of C# projects

We will be using Console App for its simplicity – but what are .NET Core and .NET Framework? It will be explained in next slide

Framework version – default should be OK

Source control option – should learn this on your own if there's not enough time to cover



Name of the solution and project – the difference will be explained in other slides

Parent folder for the solution (each solution will have its own folder within this parent folder)

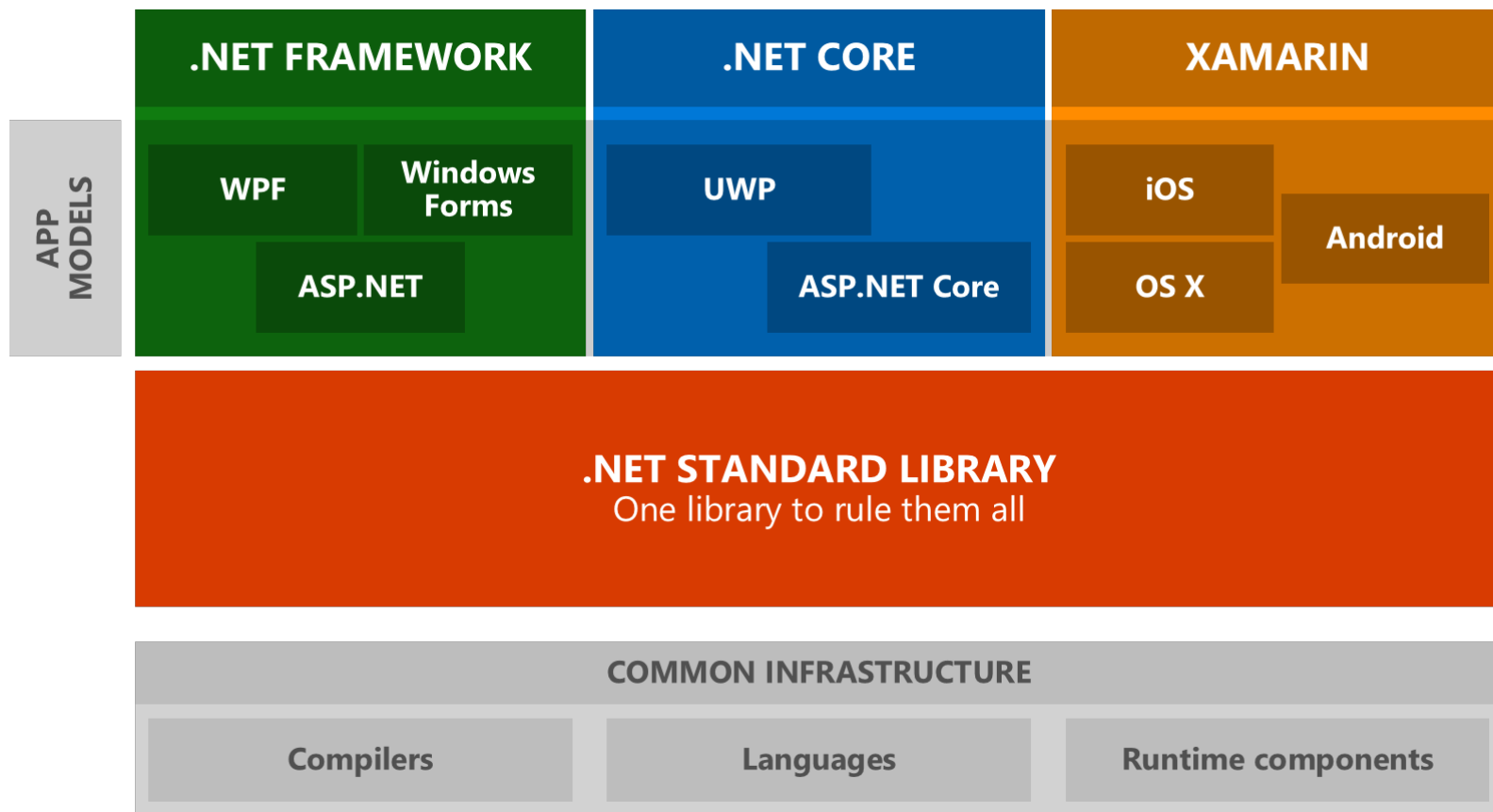
# Different Implementation of .NET

- .NET Core vs .NET Framework
  - .NET Framework
    - Used to build application that runs on Windows desktop and servers
    - Used to be the only choice
  - .NET Core
    - A flavour of .NET Framework that is cross-platform and can run on Windows, Linux and macOS
    - Contains less class libraries than .NET Framework, the standard set of class libraries for .NET Core is called as .NET Standard

<https://msdn.microsoft.com/en-us/magazine/mt842506.aspx>



# .NET Standard



<https://blogs.msdn.microsoft.com/dotnet/2016/09/26/introducing-net-standard/>

# Solution and Project

- A VS solution can contains multiple projects
- Example:
  - A company wants to have a utility that has two kinds of interface: windows desktop application and console application
  - Since the functionality are similar, we want both application to share as much implementation as possible (this is called as reusability or code reuse)
  - So we can have a VS solution with 3 projects:
    - Windows Forms App – for the windows desktop application
    - Console App – for the console application
    - Class Library – for the reusable functionality shared by both applications
- In VS, you create a solution by creating the first project in the solution

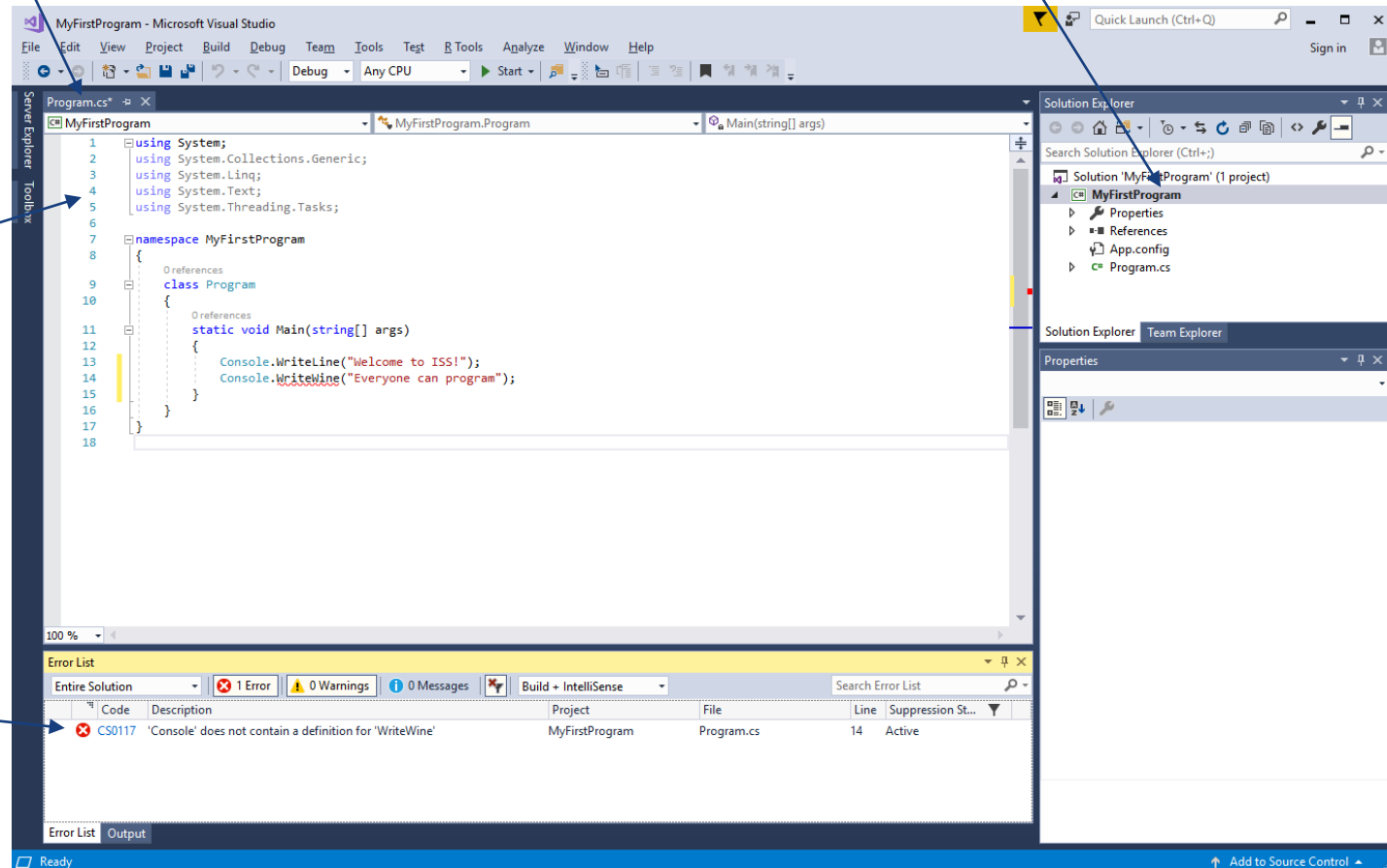
# Writing your codes

Program.cs is automatically created for you based on the template. The file is safe to be deleted

Solution Explorer – browse through the projects, folders and files in your solution

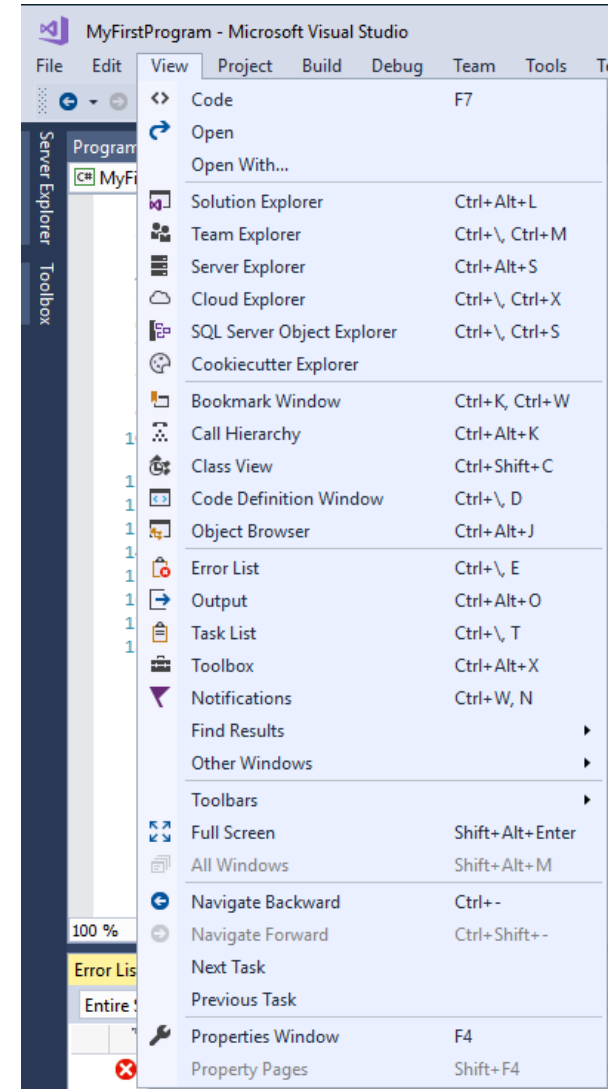
Main Editor Window – if you open multiple files, they will be shown as tabs inside this window

List of syntax errors – click on the row to go to the location shown



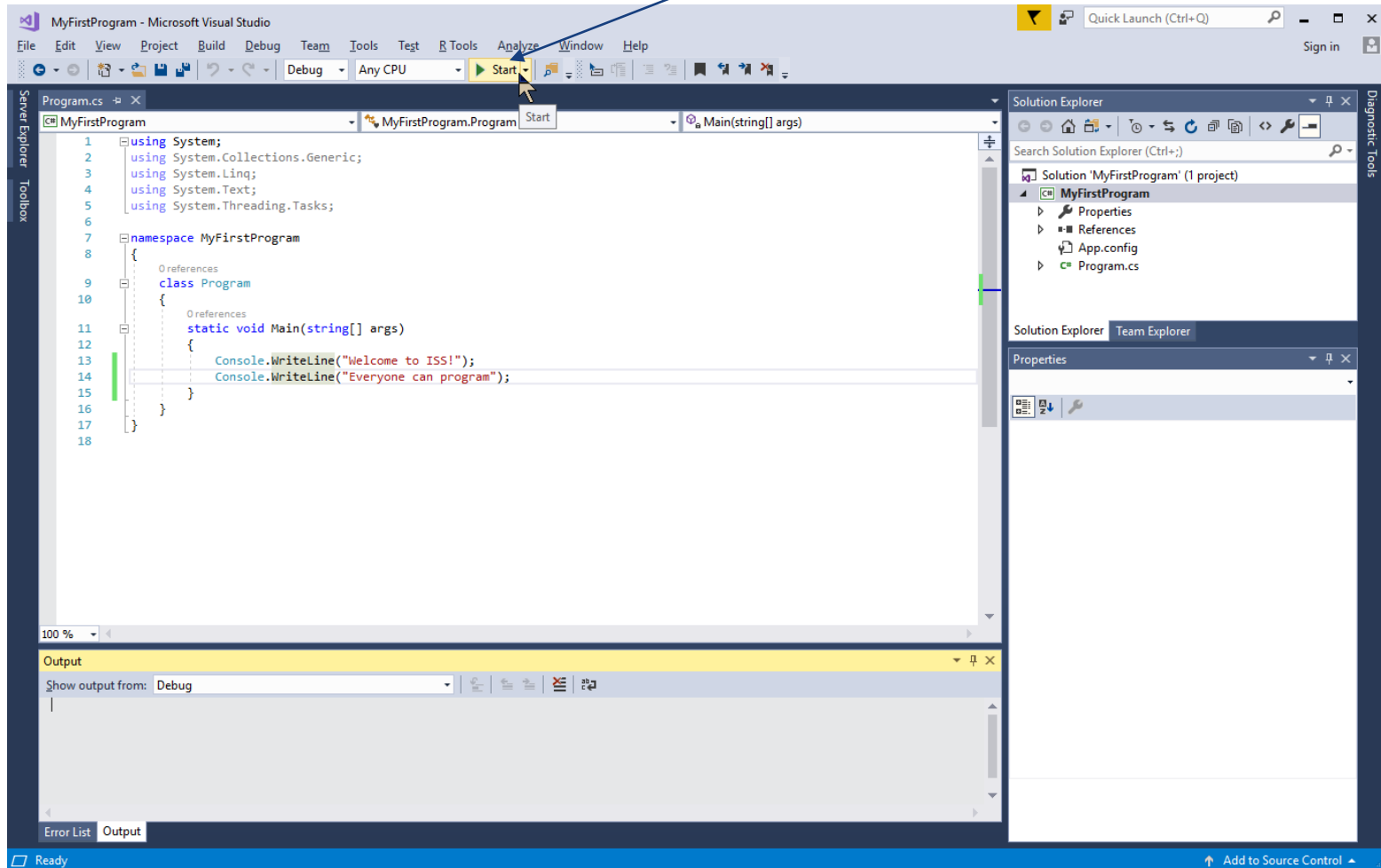
# Windows in VS

- Each of the windows have its own purpose
- You can close or hide windows that you don't use
- You can re-open the windows using the View menu items
  - It's useful to pay attention to the names of windows so that you can re-open them



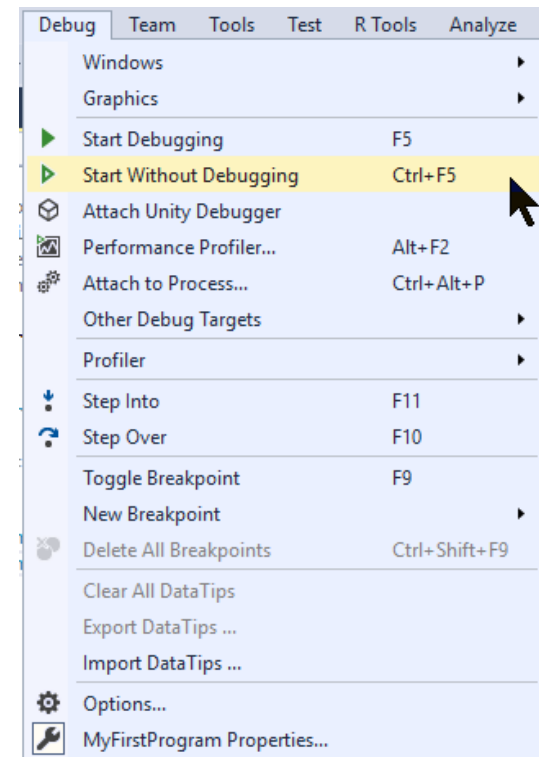
# Running your program

Once you have no syntax error, you can click on Start button to run your program – but you probably don't see the expected output – Why?



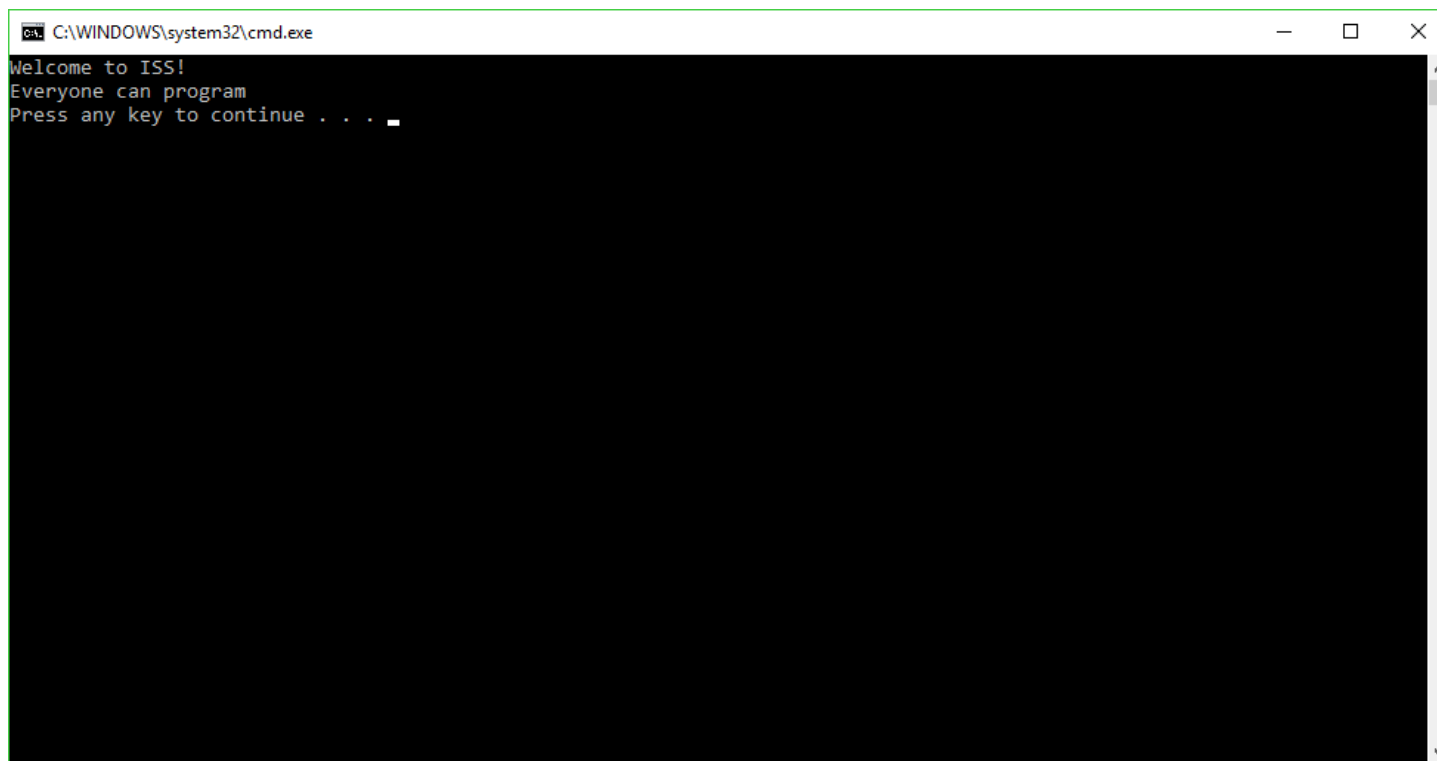
# Running your program

- There are two ways to run your program
  - Start (with) Debugging
    - This is what happened when you click the Start button
    - The program will stop when a breakpoint is reached (but we have none so far)
    - At the end of the program, the console window will be closed automatically
      - This is why we can't see our output
  - Start without Debugging
    - Breakpoints will be ignored
    - The console window will not be closed automatically
      - This is perfect for us.



# Running our program

Choose Start without Debugging or press Ctrl-F5 and we will see the output of our program



```
C:\WINDOWS\system32\cmd.exe
Welcome to ISS!
Everyone can program
Press any key to continue . . .
```

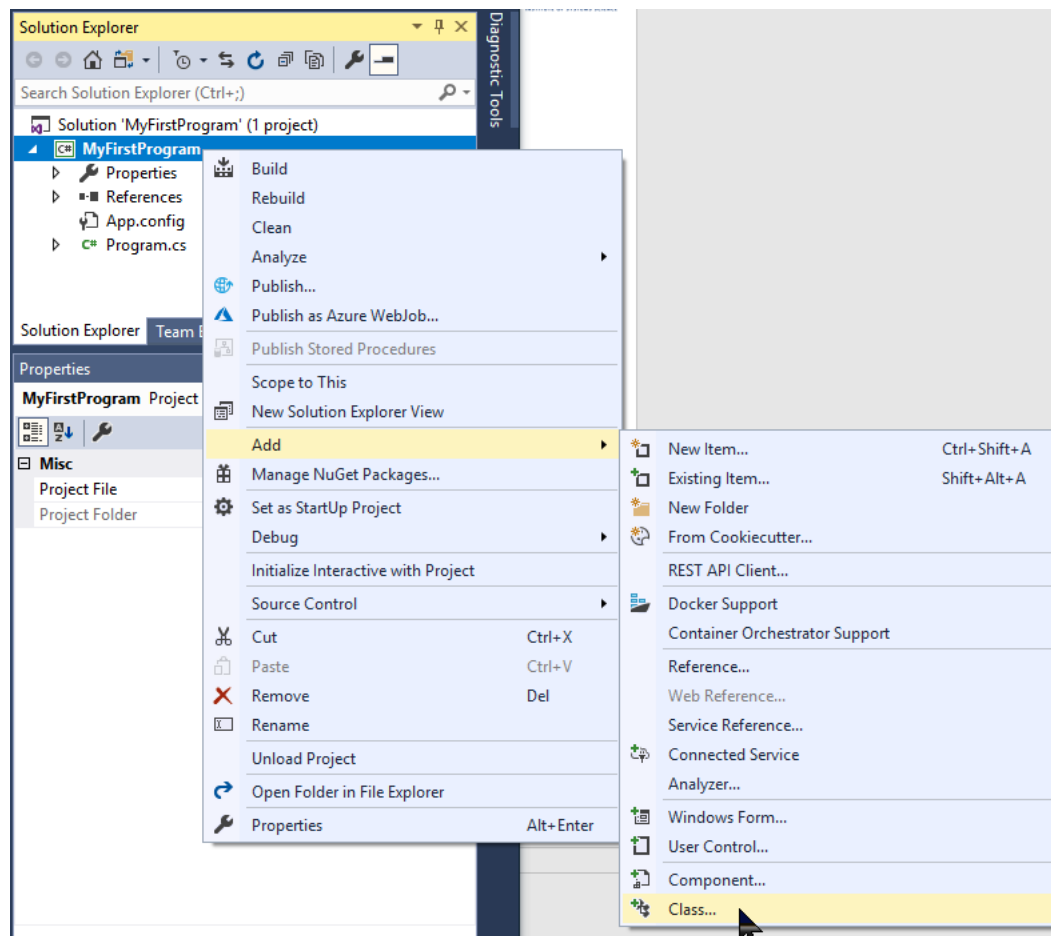
# Writing another program

- If we want to write a second program, we have at least two options:
  - Create a new solution, a new project and a new class with a `Main` method for our second program
    - Preferred approach when we are writing a totally different program and the program is relatively complex
  - Write the new program in the same project
    - Challenge: a program can have one entry point but both program have its own entry points
      - we'll learn how to solve this
    - This is probably preferred for practice so that you won't have too many folders and source files created on your hard disk.

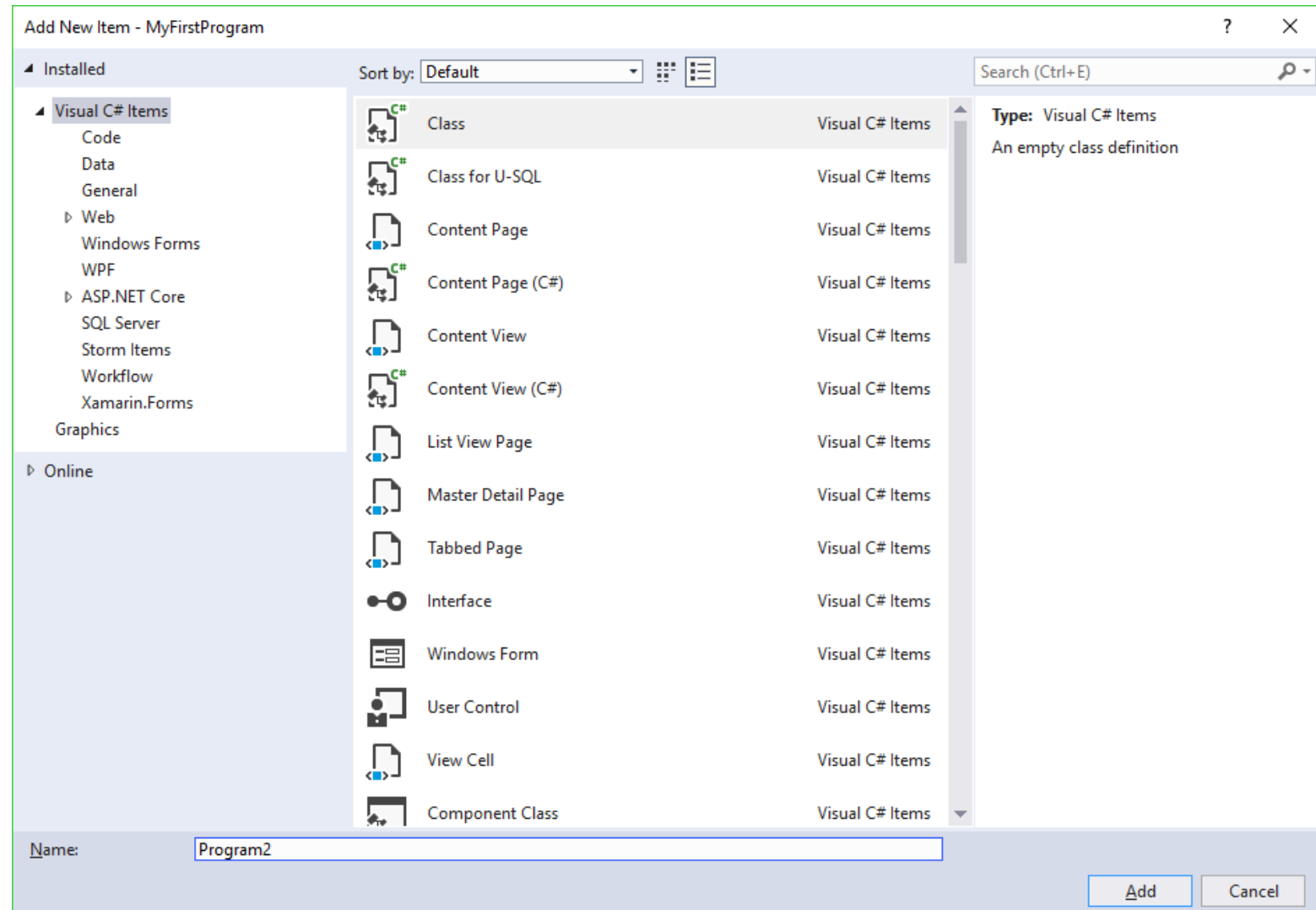


# Writing another program

- Create a new class
- Right-click on the project in Solution Explorer, choose Add > Class
- Name the class

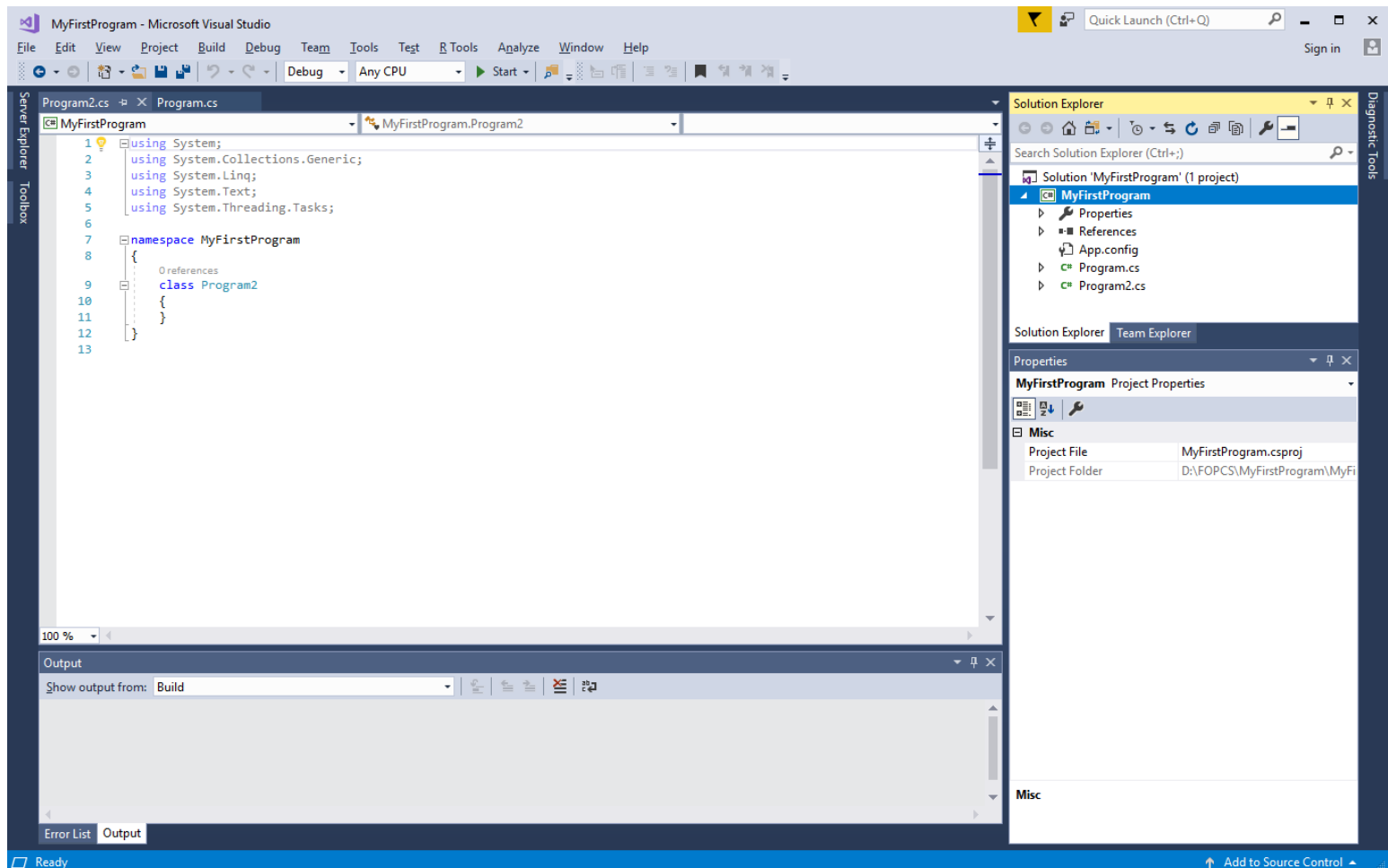


# Writing another program



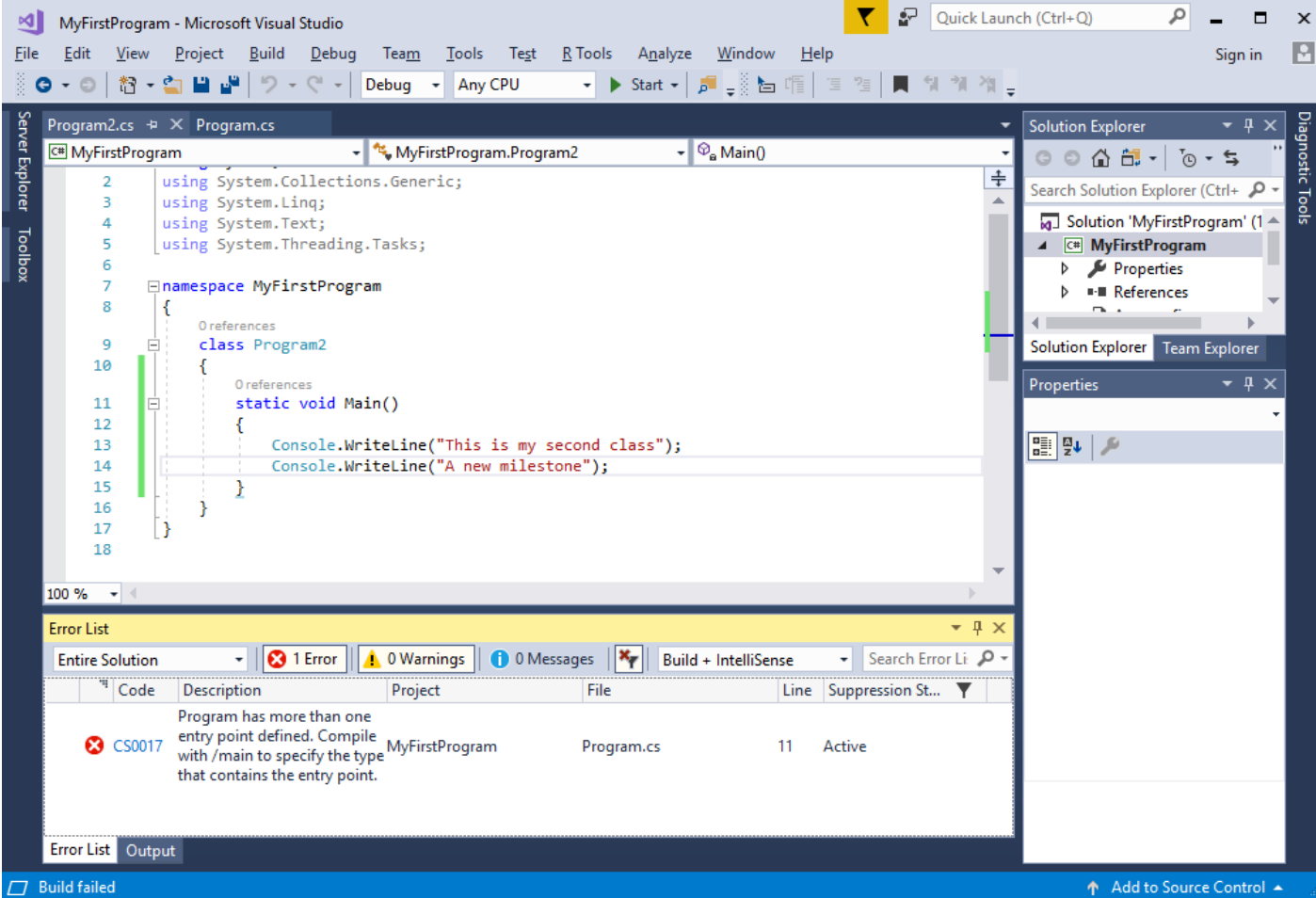
# Writing another program

The second class does not automatically have the Main method defined because normally one project will only have one Main method. So we have to write our own Main method.



# Writing another program

If we run our program, we will get this error because we have more than one possible entry points in our program. VS expects us to explicitly specify which one to use.



The screenshot shows the Microsoft Visual Studio IDE with a C# project named 'MyFirstProgram'. The code in 'Program.cs' defines a namespace 'MyFirstProgram' containing a class 'Program2' with a static 'Main()' method. The method body contains two 'Console.WriteLine' statements. The Solution Explorer on the right shows the project structure. The Error List at the bottom displays a compilation error (CS0017) indicating that the program has more than one entry point defined.

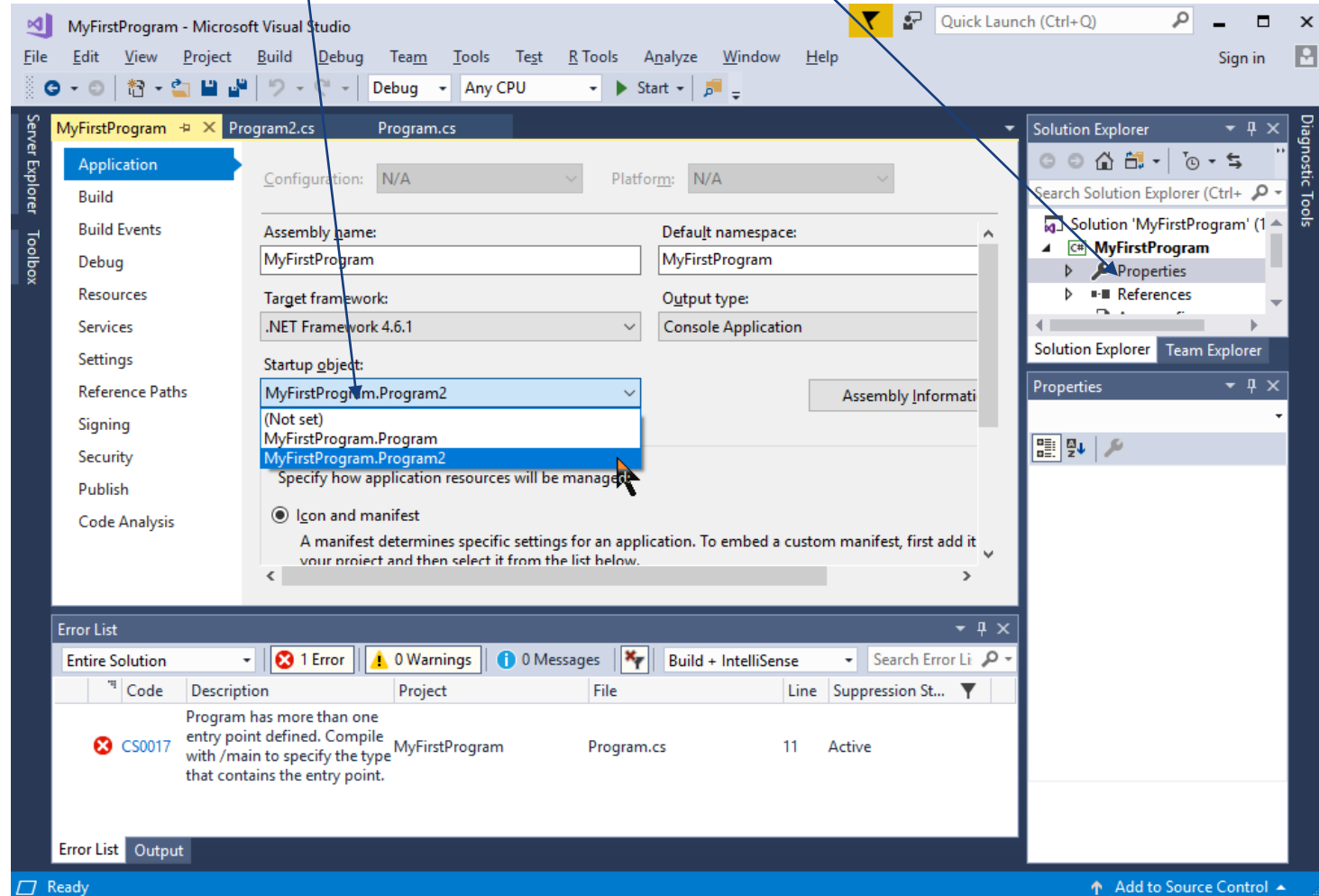
**Error List**

Code	Description	Project	File	Line	Suppression St...
CS0017	Program has more than one entry point defined. Compile with /main to specify the type that contains the entry point.	MyFirstProgram	Program.cs	11	Active

Build failed

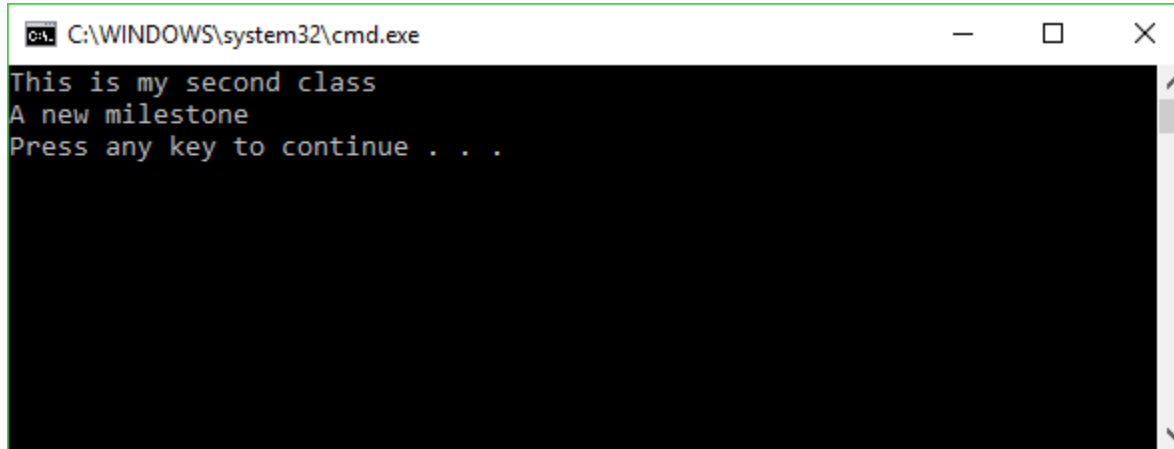
# Writing another program

Go to project properties page and nominate a class to be our startup object.  
If your new class doesn't appear, check for typo and re-build



# Writing another program

At this point we should be able to run our second program.



```
C:\WINDOWS\system32\cmd.exe
This is my second class
A new milestone
Press any key to continue . . .
```

Do we have to keep changing the startup objects?  
Yes, for now.

Once we learn conditionals, we can run different programs  
based on user's input.

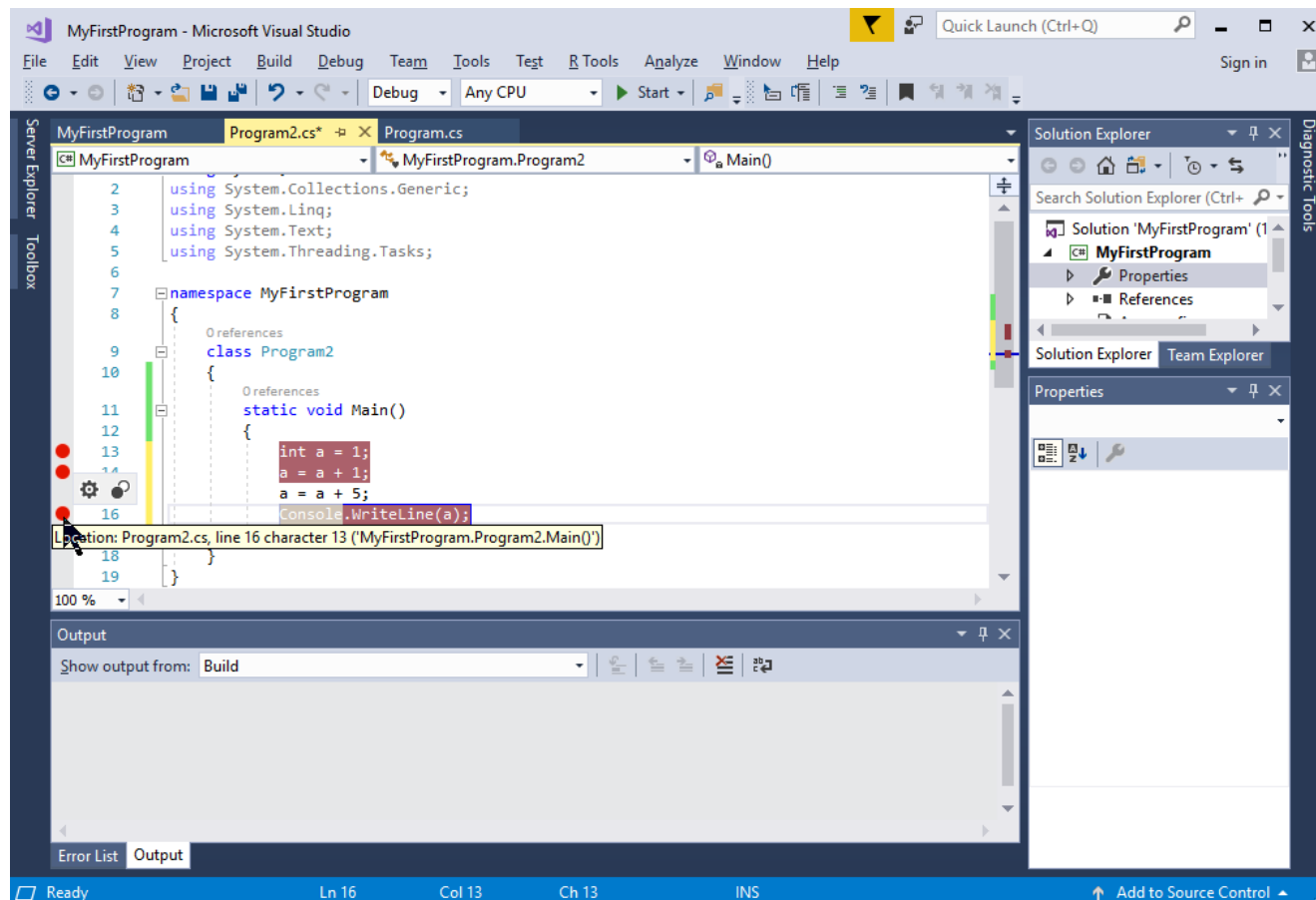
# Debugging Features

- Breakpoint
- Watch Window
- Immediate Window
- Auto Window

# Breakpoints

We can set breakpoints by clicking on the grey left border in our editor windows. Breakpoints are marked with the red circles.

Setting breakpoint means that we want our program to pause at that point (“take a break”) so that we can inspect the state of our program.





# Debugging with breakpoints

MyFirstProgram (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test R Tools Analyze Window Help

Process: [33632] MyFirstProgram.exe Thread: [15336] Main Thread

MyFirstProgram Program2.cs Program.cs

MyFirstProgram Program2 Main()

```

2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MyFirstProgram
8 {
9     class Program2
10     {
11         static void Main()
12         {
13             int a = 1;
14             a = a + 1;
15             a = a + 5;
16             Console.WriteLine(a);
17         }
18     }
19 }

```

100 %

Autos

Name	Value	Type
a	0	int

Call Stack

Name	Lang
MyFirstProgram.exe!MyFirstProgram.Program2.Main() Line 13	C#

Call Stack Breakpoints Exception Setti... Command Wi... Immediate Wi... Output

↑ Add to Source Control

Diagnostics Tools

Diagnostics session: 0 seconds (53 ms selected)

CPU (% of all processors)

Summary Events Memory Usage CPU Usage

Events

Show Events (1 of 1)

Memory Usage

Take Snapshot

Team Explorer

Quick Launch (Ctrl+Q)

We are about to execute this line

We can click on continue button to continue to the next breakpoint

Auto window shows the state of variables that are related to our current line of code

# Debugging with breakpoints

MyFirstProgram (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug

Process: [33632] MyFirstProgram.exe

MyFirstProgram Program2.cs Program.cs

MyFirstProgram

```

2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MyFirstProgram
8 {
9     //References
10    class Program2
11    {
12        //References
13        static void Main()
14        {
15            int a = 1;
16            a = a + 1;
17            a = a + 5;
18            Console.WriteLine(a);
19        }
20    }
21 }

```

Autos

Name	Value
a	0

Autos Locals Watch 1

Ready

MyFirstProgram (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools

Process: [33632] MyFirstProgram.exe Lifecycle Events

MyFirstProgram Program2.cs Program.cs

MyFirstProgram

```

2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MyFirstProgram
8 {
9     //References
10    class Program2
11    {
12        //References
13        static void Main()
14        {
15            int a = 1;
16            a = a + 1;
17            a = a + 5;
18            Console.WriteLine(a);
19        }
20    }
21 }

```

Autos

Name	Value
a	1

Autos Locals Watch 1

Ready Ln 14 Col 1

MyFirstProgram (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test R Tools

Process: [33632] MyFirstProgram.exe Any CPU Lifecycle Events Thread: [15]

MyFirstProgram Program2.cs Program.cs

MyFirstProgram

```

2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MyFirstProgram
8 {
9     //References
10    class Program2
11    {
12        //References
13        static void Main()
14        {
15            int a = 1;
16            a = a + 1;
17            a = a + 5;
18            Console.WriteLine(a);
19        }
20    }
21 }

```

Autos

Name	Value	Type
a	7	int

Autos Locals Watch 1

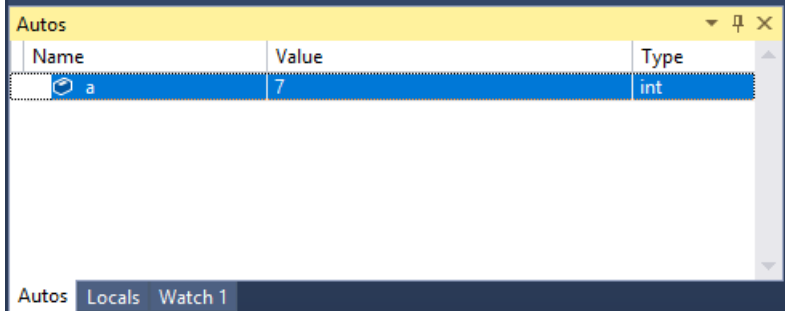
Ready Ln 16 Col 13 Ch 13

# Options on a breakpoint

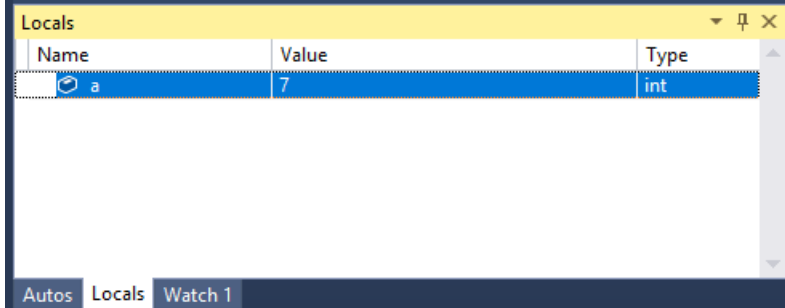
- Continue to the next breakpoint
- Continue to the next line (Debug > Step Over)
- Continue into the implementation of the method on our current line (Debug > Step Into)
  - Useful when we have learned about methods
- Continue to the next line outside of our method implementation (Debug > Step Out)
  - Step Into followed with Step Out = Step Over

# Monitoring Program State

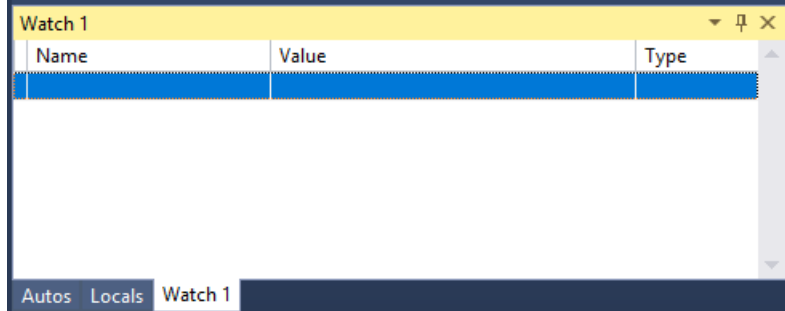
- With IDE, we can check the value of our variables
- Auto window
  - Display the state of variables related to the current line of code
- Watch window
  - Allow user to manually enter the variables to be monitored
- Local window
  - The state of all local variables



Name	Value	Type
a	7	int

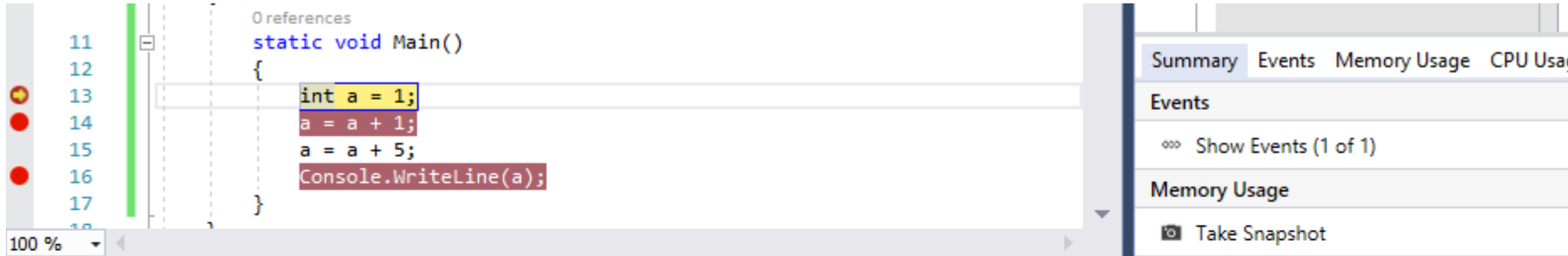


Name	Value	Type
a	7	int



Name	Value	Type
------	-------	------

# Immediate Window



```

11
12
13  int a = 1;
14  a = a + 1;
15  a = a + 5;
16  Console.WriteLine(a);
17
  
```

100 %

Name	Value	Type
a	0	int

Summary Events Memory Usage CPU Usage

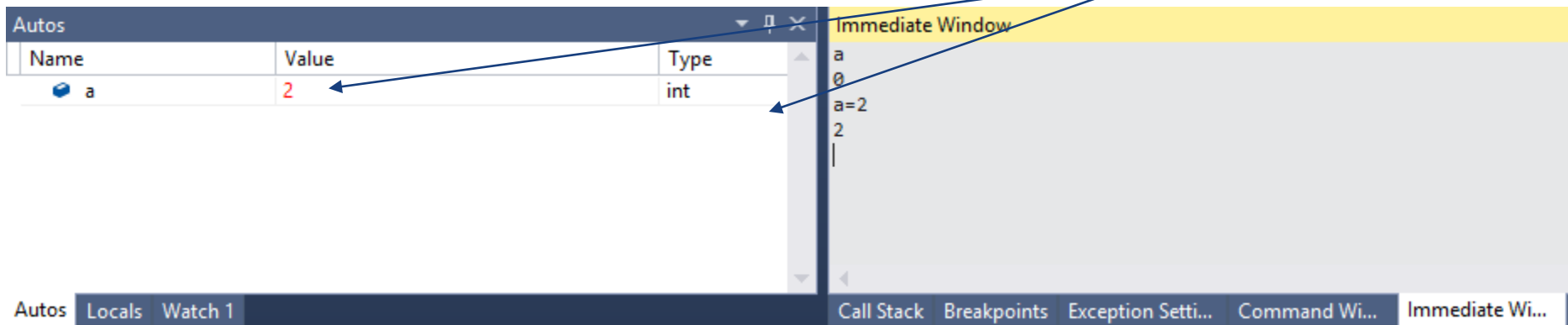
Events

Show Events (1 of 1)

Memory Usage

Take Snapshot

Allow you to perform C# statements, including changing the variable value during a break



Name	Value	Type
a	2	int

Immediate Window

```

a
0
a=2
2
|
  
```

Autos Locals Watch 1

Call Stack Breakpoints Exception Setti... Command Wi... Immediate Wi...

# Debugging a Program

- When program doesn't run as expected (a program bug), we need to investigate the problem (debug).
- Many approach to debugging:
  - Write (log) variable values to screens or files
  - Use breakpoint and step through your program
  - Comment out part of your codes and check how your program runs without those part
    - Try to isolate the bug

# Intellisense

- What is Intellisense?
  - It is a editor feature, that helps the developer with prompts, lookup and other visual features.
  - It improves the programmers editing productivity
- What are the features?
  - List Members
  - Parameter Info
  - Quick Info
  - Complete Word
  - Automatic Brace Matching

<https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2017>

# Intellisense

```

// Place the frame in the current Window
Window::Current->Content = rootFrame;

Window::C
// Ensur
Window::
  }, task_cont
}
else
{

```

Close  
 Closed  
 Content  
 CoreWindow  
 Current

public : void Windows::UI::Xaml::IWindow::Close()  
 File: Windows.winmd  
 + 1 overload

```

InitializeComponent();
string s = "hello";
bool b = s.EndsWith("o", |)

```

▲ 2 of 3 ▼ **bool string.EndsWith(string value, StringComparison comparisonType)**  
 Determines whether the end of this string instance matches the specified string when compared using the specified comparison option.  
**comparisonType:** One of the enumeration values that determines how this string and value are compared.

```

InitializedComponent();
string s = "hello";
bool b = s.EndsWith(

```

bool string.EndsWith(string value) (+ 2 overloads)  
 Determines whether the end of this string instance matches the specified string.  
 No overload for method 'EndsWith' takes 0 arguments



# Summary

- IDE is a very useful productivity tools for programmers/ developers
  - Provide templates
  - Editing features
  - Automate application building
  - Debugging features
- Not discussed
  - Source control features
  - Deployment features
  - Can install plugins to allow for additional features